Research Paper

Volume : 2 | Issue : 2 | Feb 2013 • ISSN No 2277 - 8179

Comparison Of Conventional Approach with Component Based Software Development

Engineering

KEYWORDS : Component based Software Engineering, New Approach, software engineering.

Gottipalla Ashok Kumar Assistant Professor, Department of CSE, Krishna Murthy Institute of Technology and Engineering, Hyderabad

ABSTRACT

Strongyloides stercoralis infection is widely prevalent all around the world. Risk factors for strongyloides infection include HIV, malignancy, immunosuppressive therapy, diabetes and malnutrition .Infection can result in asymptomatic chronic disease of the gut, which can remain undetected for decade but in immunocompromised patients specially those receiving long-term corticosteroid therapy, hyperinfection can occur, resulting in high mortality rates (up to 87%).

One year data was analysed to study the Strongyloides stercoralis infections, clinical presentations, treatment and the outcome in patients attending a tertiary care hospital.

Strongyloides stercoralis was diagnosed in 7 patients. Three of these patients were on corticosteroid therapy and two were malnourished, others apparently did not have any high risk factor for the infection. All were HIV negative. Corticosteroid therapy is an important risk factor for Strongyloides stercoralis infection. Strongyloides Co infection with other helminths can occur so it should always be kept in mind. Hyperinfection with Strongyloides stercoralis results in high mortality despite of treatment.

INTRODUCTION

Component Based Software Development (CBSD) is a branch of software engineering that emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. It is a reuse-based approach to defining, implementing and composing loosely coupled independent components into systems. This practice aims to bring about an equally wide-ranging degree of benefits in both the short-term and the long-term for the software itself and for organizations that sponsor such software.

Main goal of component-based development is to build and maintain software systems by using existing software components. They must interact with each other in system architecture. Moreover CBSE aims to improve the Quality (Flexibility, Maintainability, Reliability) of software systems and productivity & Time-to-market of software development through enabling the easy assembly of software from existing building blocks. Other reasons can be seen by comparing the conventional design and development with Component Based Software Development. Below Table 1 shows the comparisons between conventional design and CBSD.

Table 1: The comparisons between conventional design and CBSD

Area of Difference	Conventional Approach	CBSD
Architecture- How the system is setup	Monolithic	Modular
Components- the pieces of the system	Implementation and White Box (Programmer is given access to the source code)	Interface and Black Box (Components can not be adapted or changed)
Process- How the system is put together	Big Bang & Waterfall (Analysis to design to implementation to testing)	Evolution and concurrent engineering (Component development to component integration)
Methodology-How the system changes through time	Built from scratch	Composition
Organization-Market for buying and selling components	None	Specialized-Component vendors and integrators

From the above table1, it is observed that, In the context of software architecture ,conventional approach adopts monolithic development whereas CBSD adopts a Modular approach. The development process followed in conventional approach is generally waterfall model which goes from analysis to testing one phase at a time, whereas in case of CBSD a more robust evolutionary and concurrent engineering model is adopted there is more emphasis on component development and component integration. Another level where conventional methodology differs from CBSD is the methodology. The conventional software are developed from scratch whereas components are composed together to make a new product in case of CBSD. Another advantage is that in case of CBSD there are specialized vendors and integrators for buying and selling the components.

The Y Software Life Cycle Model:

The Y model, (Fig. 1), has been proposed as a viable alternative to address software reusability during component-based software production. The creation of software is characterized by change and instability, hence the diagrammatic representation of the Y model considers overlapping and iteration where appropriate. Although the main phases may overlap each other and iteration is allowed, the planned phases are: domain engineering, frame working, assembly, archiving, system analysis, design, implementation, testing, deployment and maintenance. The main characteristic of this software life cycle model is the emphasis on reusability during software creation and evolution and the production of potentially reusable components that are meant to be useful in future software projects. Reusability implies the use of composition techniques during software development; this is achieved by initially selecting reusable components and assembling them, or by adapting the software to a point where it is possible to pick out components from a reusable library. It is also achieved through inheritance, which is naturally supported by an object-oriented approach. Reusability within this life cycle is smoother and more effective than within the traditional models because it integrates at its core the concern for reuse and the mechanisms to achieve it.

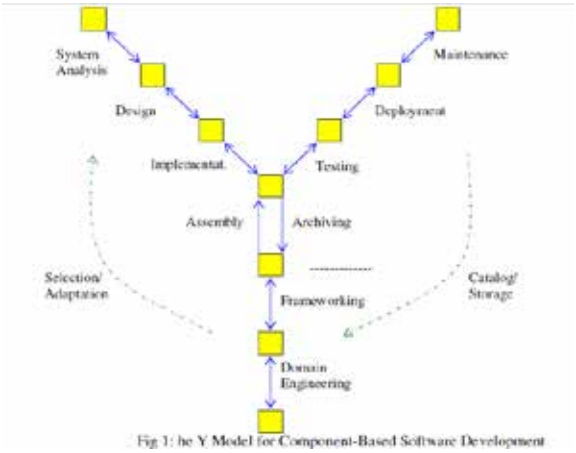

Fig 1: he Y Model for Component-Based Software Development

Ensuring Quality in Component Based Software Engineering

Quality is a very important aspect of software development and in component based software engineering there are additional considerations involved. First of all quality needs to be incorporated into the components chosen for the final system. The quality characteristics of components that need to be looked at are: functionality, interface, usability, testability, maintainability, and reliability . Some software metrics that can be used to classify components are: size, complexity, reuses frequency, and reliability. Size can affect reuse cost and quality. If a component is too small the benefits will not exceed the cost of managing it and if a component is too large that it is difficult to have high quality. Complexity also affects reuse and quality, a simple or trivial component is not beneficial for reuse while an overly complex component is difficult to inherit high quality from. The number of times that a component is used is a very useful indicator of how useful a component is. The reliability of component is important as it will show the probability of failure-free operations. There are many standards that software components can follow which include but are not limited to: ISO9001 and CMMI. A big focus is put on the analysis of the components in a component based software engineering system as most of the system is made up of components.

Advantages and Disadvantages of CBSD

The following reasons are some of the significant advantages that component based development offers. The first being independent extensions. Legacy software is plagued by a lack of flexibility, components are units of extension and the component model describes exactly how an extension can be made. The component model allows easy extensions for systems. Another advantage of component based development is the component market. The component models provide standards that independently developed components can follow and this will provide large system developers with fewer unanticipated interactions between software components. This allows independent developers to develop components for niche markets. A big advantage of component based software development is the reduced time to market. Using already developed components means less design, coding, and testing which in turn lowers production time as well as production costs.

As with everything there are some disadvantages to component based development. The time and effort it takes to develop a software component is large and thus the cost to develop a component can be high. The software components may have more functionality than a developer may need for a project that they are working on. In some cases a component may have to be used up to 5 times before the cost of purchasing a component can be justified or recovered. Another disadvantage is that requirements in some component based technologies are lacking, however with the three current big technologies it is becoming less of a problem. There also could be a conflict between the usability and reusability aspects of components. Components need to be general and adaptable which leads to components costs that are more expensive and more complicated to use. The last disadvantage of CBSE is that the maintenance cost decreases for the application but can increase for the component, depending on the complexity of the encompassing system. The separate life-cycles together with different requirements for the application and component could be problematic.

Conclusion

Component based software engineering is a new paradigm of industry as well as very interesting and useful software development model. It provides an easy way to assemble software using components that have been previously developed. CBSD shifts the emphasis from programming software to composing software systems. It offers a quick and cheap way to develop software systems. As its popularity grows the market for already development software components grows and offers an interesting direction for software evolution. In order for this type of model to see success there is a great need for well defined standards for components. These standards would make it possible to assume quality for software components and make it easy for large system developers to fit components into their software frameworks. It is an exciting technology and it will be interesting to see it progress in the future.

REFERENCE

[1] Luiz Fernando Capretz ., Y: A New Component-Based Software Life Cycle Model, Journal of Computer Science 1 (1): ISSN 1549-3636, 76-82, 2005. | [2] Blevins D., Overview of the Enterprise JavaBeans Component Model, in G. T. Heineman and W. T. Councill (editors), Component-Based Software Engineering: Putting the Pieces Together, Addison Wesley, 2001. | [3] Hoek A., Rakic M., Roshandel R., and Medvidovic N., Taming architecture evolution. In Proceedings of the 6th European Software Engineering Conference (ESEC) and the 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9), 2001. | [4] Crnkovic, I. and Larsson, M. A Case Study: Demands on Component-based Development, Proceedings 22nd International Conference on Software Engineering, ACM Press, 2000. | [5] Garlan D., Monroe R.T. and Wile D., Acme: Architectural description of component-based systems. In G.T. Leavens and M. Sitaraman, editors, Foundations of Component-Based Systems, pages 47–68. Cambridge University Press, 2000. | [6] Medvidovic N. and Taylor R.N., A classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering, 26(1):70–93, 2000. | [7] Kamiya, T., Kusumoto S., Inoue K., Mohri Y., Empirical Evaluation of Reuse Sensitiveness of Complexity Metrics, pages 297-305, Information and Software Technology Journal, 1999,. | [8] Schneider J.G. and Nierstrasz O., Components, scripts and glue. In L. Barroca, J. Hall, and P. Hall, editors, Software Architectures Advances and Applications, pages 13 – 25. Springer, 1999. | [9] Szyperski C., Component Software –Beyond Object-Oriented Programming. Addison-Wesley, 1998. | [10] Szyiperski C. and Pfister C., Workshop on Component-Oriented Programming, Summary. In Muhlhauser M. (ed.) Special Issues in Object-Oriented Programming – ECOOP96 Workshop Reader, Springer 1997 . | [11] Aaron J Miller, www.uwplatt.edu/.../ComponentBasedDevelopment_milleraar.docx | [12] Shaw M. and Garlan D., Software Architectures: Perspectives of an Emerging Discipline. Prentice Hall, 1996. |